AUSTRALIAN OS9 NEWSLETTER

EDITOR	Gordon Bentzen	(07) 344-3881
SUB-EDITOR	Bob Devries	(07) 372-7816
TREASURER	Don Berrie	(07) 375-3236
LIBRARIAN	Jean-Pierre Jacquet	(07) 372-4675
SUPPORT	Brisbane OS9 Users Group	

Addresses for Correspondence

Editorial Material:

Gordon Bentzen
8 Odin Street
SUNNYBANK Qld 4109

Subscriptions & Library Requests:

Jean-Pierre Jacquet
27 Hampton Street
DURACK Qld 4077

Volume 5 February 1991 Number 1

AUSTRALIAN OS9 NEWSLETTER
Newsletter of the National OS9 User Group
Volume 5 Number 1

EDITOR : Gordon Bentzen
SUBEDITOR : Bob Devries
TREASURER : Don Berrie
LIBRARIAN : Jean-Pierre Jacquet
SUPPORT : Brisbane OS9 Level 2 Users Group.

Many OS9'ers around the world are looking more to OS9 68K as the future for OS9. The new hardware offerings from Frank Hogg Laboratories, Tom-Cat etc and Interactive Media Systems' MM/1, have no doubt increased the interest in OSK. A third company has now entered the scene offering a MC68000 based machine. Delmar Co. is advertising their System IV computer. Their System IV was presented at the Atlanta CocoFest.

The other point of significance is of course the decision by Microware to drop support of 6809 OS9. Now all we need is a realistic price from Microware for the OS9 68K operating system. It does seem that the three companies mentioned above must have negotiated a volume licence with Microware as the MM/1 and System IV machines, according to the ads, come complete with Professional OSK. We also hear that Microware will no longer support Personal OSK, so it seems that they are only interested in the commercial users, and as far as we can tell, are not too interested in the personal or hobby user.

The good news is that the future for the OS9 operating system looks brighter than ever, even in the hobbyist's arena, thanks to the efforts of IMS, Frank Hogg and now Delmar.

A new European OS-9 User Group has recently been established, prompted mainly by the new entrants, IMS, Frank-Hogg, and Delmar, in the OSK systems. The European OS-9 User Group has existed for some time and supported 6809 OS-9 in much the same way the Australian OS9 User Group has. A good deal of our expertise, if we can call it that, is based on CoCo OS9. (Microware's version of OS9 for the Tandy Colour Computer). It does seem that the "NEW" European OS-9 User Group plans to move heavily into OSK, and that is no doubt the way into the future. This user group publishes a newsletter on disk with all the hard work being done by Peter Tutelaers and Burghard Kinzel.

As the National OS9 User Group we have a reciprocal membership arrangement with the European group and exchange newsletters ,on disk, on a regular basis. We have legal distribution rights to their public domain library, and they have similar rights. Should you wish to join the European group as an individual member please contact us for details. We are able to provide P.D. programmes etc. from here for the normal copy charge of $2.00 per disk plus return postage. Please send your requests for P.D. disks to our librarian.

NEWSLETTER Volume 5

You may have noted that this first edition for 1991 is also the beginning of Volume 5, and this is an achievement in which we take some pride. I don't think that Bob, Don or myself anticipated this when we first decided to resurrect the National OS9 User Group and produce a monthly newsletter from sunny Queensland. The monthly presentation of interesting articles does become difficult at times, and we really need some input from other members. So let's make 1991 our best year yet for member contributions. Send us your articles, reviews, comments or questions, preferably on disk as a "vanilla" ASCII file. We can read any OS9 disk format in 5.25" or 3.5" disks, or even MSDOS, RSDOS format if you must.

Cheers, Gordon.

oooooooooo0000000000oooooooooo

CoCo-Link

CoCo-Link is an excellent magazine to help you with the RSDOS side of the Colour Computer. It is a bi-monthly magazine published by Mr. Robbie Dalzell. Send your subscriptions to:

CoCo-Link
31 Nedlands Crescent
Pt. Noarlunga Sth.
South Australia
Phone: (08) 3861647

Patch Mfree for 1 Meg
By Bob Devries

I have for some months now had a CRC/Disto 1 Meg upgrade. It works very well indeed, and although our version of OS9 does not make full use of it yet, perhaps future patches will do so. Technically, it really requires skilled soldering work to install, because it requires installing a header onto the 68B09 CPU chip. In the Australian CoCo 3 it is quite a tight fit, but with a bit of juggling it goes in OK. It requires a 9 volt 1 Amp plug-pak power supply to power the extra RAM chips, and this is not supplied with units sold to Australia because of the voltage and frequency difference in the mains power. Tandy sells a suitable unit as catalogue number 273-9651, which has a switchable output and multiple plug types. One thing could worry some of you though, it runs very hot! As I am writing this, it is 35 degrees C in my lounge, and the top of my computer, where the extra RAM chips are is almost too hot to touch. Apart from slightly bending the plastic in that spot, however, the upgrade has not given any trouble.

Now for the software run-down. It would appear that everything runs with the 1 Meg installed, except for a few games that 'cheat' and directly manipulate the hardware. However, unless the supplied programme 'mega' is run (preferable first-up in the startup file), OS9 does not know about the upgrade, and all runs as usual. Once the 'mega' programme is run once (it does not need to be in the shell, as it is a once-off programme), OS9 will be able to use the extra 512k of RAM. The limitation seems to be that the system memory map is still limited to that area in the first 64k map, and so you can NOT run more programmes. You can, however, open more windows, including graphics ones. This means you may be able to multi-task more graphics programmes such as picture viewers and so on.

One of the programmes which did not seem to work correctly was 'Mfree'. I did not notice this myself, until one of the members of the Brisbane usergroup pointed it out to me on his computer. This is what it did on his machine:-

```
Blk Begin   End    Blks  Size
--- ------ ------ ---- ------
 25  4A000  7BFFF   19   200k
 40  80000                    ==== ======
          Total:    58   704k
```

As you can see, the last part of the fourth line is not finished. Oddly enough, it did not do this on my set-up at all. After much head shaking and investigation, I found that it would only play up if the last 8k memory block was not being used at the time 'Mfree' was run. Of course, this to me is like waving a red flag at a bull, and I HAD to find a solution. I disassembled 'Mfree', and after a few hours poring over the result, I found that the 'Mfree' command (rightly) assumes that the top memory block available on the computer was in use, but because it was in fact not like that, it 'stuffed up'.

I say rightly, because when OS9 starts up, it checks for what it thinks is the top of RAM, and places the kernel (OS9p1, INIT, etc) there. However, because OS9 is informed of the increase in memory only after it has started up, it does not put the kernel in block $7F as it should, but in the old 512k top-of-RAM, which is block $3F. Hence, the top block remains unused until the second window screen is opened. That is, Term first, then, in my system, W7, and then W1 which is assigned to the top memory block by Grfdrv. The difference between my system and the other, was that I always started my system with at least three windows, and the other with only two.

Anyway, what I have done, is patched 'Mfree', written a file to be used by Bob Santy's 'Ipatch', and to enable me to give you all a copy of the patch file, I have written a Basic09 utility which creates the 'Mfree.ipc' file. All you who have a Disto 1 meg upgrade, already have the programme 'Ipatch', it is on the disk which came with the upgrade. For those of you who want to do this patch anyway, and do not have a 1 meg upgrade yet, if you have 'Ipatch' you can do the patch, if not, it is available from our PD software library in the usual way.

Now without boring you any further, here is the basic09 programme:-

```
PROCEDURE mfree_patch
 0000      (* programme to create a file called 'mfree.ipc' *)
 0033      (* which may be used to patch 'Mfree' to work    *)
 0066      (* correctly with the Disto 1 Meg upgrade        *)
 0099
 009A      DIM a:INTEGER
 00A1      DIM b:BYTE
 00A8      DIM path:INTEGER
 00AF
 00B0      CREATE #path,"mfree.ipc":WRITE
 00C4
 00C5      FOR a=1 TO 179
 00D5        READ b
 00DA        PUT #path,b
 00E4      NEXT a
 00EF
 00F0      CLOSE #path
 00F6      END
 00F8
 00F9      DATA $01,$EB,$01,$F3,$02,$00,$03,$00
 011D      DATA $01,$00,$01,$EB,$F3,$02,$00,$07
 0141      DATA $00,$02,$00,$02,$81,$C2,$82,$D9
 0165      DATA $02,$00,$12,$00,$01,$00,$01,$02
 0189      DATA $03,$02,$00,$87,$00,$01,$00,$01
 01AD      DATA $E6,$EE,$02,$00,$8D,$00,$0A,$00
 01D1      DATA $0A,$F5,$17,$00,$DD,$17,$00,$EF
 01F5      DATA $17,$00,$D7,$FD,$17,$00,$E5,$17
 0219      DATA $00,$F7,$17,$00,$DF,$02,$00,$B9
 023D      DATA $00,$01,$00,$01,$4F,$57,$02,$00
 0261      DATA $BD,$00,$01,$00,$01,$65,$6D,$02
 0285      DATA $00,$D1,$00,$01,$00,$01,$61,$69
 02A9      DATA $02,$00,$DA,$00,$01,$00,$01,$2E
 02CD      DATA $04,$02,$00,$F7,$00,$01,$00,$01
 02F1      DATA $3B,$43,$02,$00,$FD,$00,$04,$00
 0315      DATA $04,$25,$17,$00,$98,$2D,$17,$00
 0339      DATA $A0,$03,$01,$06,$00,$03,$00,$0B
 035D      DATA $67,$20,$A1,$6F,$CE,$00,$00,$10
 0381      DATA $9C,$02,$27,$02,$20,$99,$02,$01
 03A5      DATA $0C,$00,$01,$00,$01,$41,$39,$02
 03C9      DATA $01,$E8,$00,$03,$00,$03,$A3,$C8
 03ED      DATA $70,$59,$47,$69,$04,$01,$EB,$00
 0411      DATA $00,$00,$00
 0421
```

Of course, it is very important to get all those bytes of data correct, so check and re-check them.

Bob Devries.

oooooooooo0000000000ooooooooooo

TUTORIAL IN 'C' PROGRAMING

INTRODUCTION

This is an introduction to the 'C' language as running on the 'CoCo'. It is expected that you are familiar with BASIC or similar languages.

The only manner in which to learn any computer language, including C, is to actually write and debug a few programs. This tutorial

will only give you a flavor of C. If you like it get OS9 and C for your CoCo and learn how to use it.

GENERAL

C was developed in the mid 1970's as an enhancement of a language called 'B', developed by AT & T in 1973, as an attempt to get some of the power back in PL/1, which is a simplification at ALGOL 60. C become so popular that UNIX was rewritten in C. C is extremely portable in the source code level. Unfortunately, there are many versions of C, but mostly all of them are using a subset of C. Microware C is right after Kernighan and Ritchie, and the only difference is in some obsolete call's and Microware has a couple of OS9 calls which not exist under UNIX. All calls are kept with the UNIX name under OS9.

DESCRIPTION

C is a "not-very-high-level" language. It generally deals with low-level concepts on the same level as macro assemblers.
Thus, for example, although it deals with characters, it does not deal with strings. This type of ruthless language simplification will probably bother most programmers already familiar with APL, BASIC, or COBOL more than any other feature of the language.

C provides the control structures required for structured programming. It requires the declaration of all variables used by the program. It provides a block-structured syntax which prevents one subroutine from interfering with the variables of another subroutine, another highly-desirable feature of many structured languages.

VARIABLE DECLARATIONS

As just noted, all variables must be declared before their use. Variables declared within subroutines are called "local" and those declared outside of subroutines are called "global". C has several types of basic variable declarations, as follows:

```
int       integer (usually 16 bits)
short int short integer (usually 8 or 16 bits)
char      single character (usually 8 bits)
float     single-precision floating point
(usually 32 bits)
double    double-precision floating point
(usually 64 bits)
loan int  long integer (usually 32 bits)
```

An array of a certain variable type is designated by placing the constant number of element in the array, in brackets, after the variable name, in the type statement.

Variable and subroutine names in C should begin with a letter and may be composed of letters and numbers. Some implementations allow underline (Microware), at(@), and dollar as additional subsequent symbols. Some distinguish between upper and lower case letters, and some do not. Some require uniqueness in the first six characters (Microware).

CONSTANTS

Type "char" constants may be entered as a single character enclosed in single quotes, or as a symbolic escape sequence of one of the following forms:

```
\n      (newline)
\t      (tab)
\b      (backspace)
\r      (carriage return)
\f      (form feed)
\l      (line feed)
\\      (backslash)
\'      (single quote)
\0      (null)
\nnn    (octal nnn)
\xnn    (hex nn)
```

Type "char" constants may be used wherever "int" constants may be used. However, various implementations vary concerning the sign-extension or truncation employed in the conversion of type "char" to type "int".

A string constant, as in BASIC, is a series of zero or more characters (or symbolic escape sequences) surrounding by double quotes.

FUNCTIONS

A C program is a series of one or more function definitions. By convention, the first (or only) function executed is named "main". The first example happens to be exactly one function:

```
main()
{
    printf ("hello, world\n");
}
```

This trivial C program is intended to print the following:

```
hello, world
```

and illustrates, by example several points about

functions. The body of "main" is enclosed in open and close braces, as required for all functions. The library function "printf" is called with one string constant argument. The function call statement is followed by a semicolon. The function "main" (as shown here) has no parameters, but if it did, they would be listed between the parenthesis, separated by a comma, and declared as local variables, as in the following function definition example:

```
add1(n)
int n;
{
    n = n + 1;
    return (n)
}
```

which also illustrates several items concerning functions. The primary point is that functions can return values; if a function does not have an explicit type, it is defaults to type "int". Technically, all C parameters are passed by value, not by address, making it more difficult to pass results back to the caller, but solving many "side-effect" problems.

Results may be returned to the caller through globals, arrays, or pointers, among other methods. However, this method of parameter passing avoids the FORTRAN horror of accidentally modifying constants, as in the case of passing a constant, not a variable, to the "add1" function.

The EXPANSIBILITY of C is partially due to the ease of definition of C functions in a private library; "printf" is one such library function.

EXPRESSIONS

Although expressions in C may look a little strange at first to someone familiar with BASIC, there are strong parallels between the languages in terms of expression formation. The arithmetic operations are as follows:

+	addition
−	subtraction
*	multiplication
/	division
%	modulus
−	unary negation
++	unary increment
−−	unary decrement
,	sequential evaluation

The logical operations are as follows:

&&	and
||	or
!	unary not

The relational operators are as follows:

>	greater
>=	not less
<	less
<=	not greater
==	equal
!=	not equal

Note that the "equal" operator is "==", not "=". The assignment operator "=" is allowed inside expressions, and has the same interpretation as outside expressions, of changing the value of the variable on its left side to that of the expression on its right. The conditional expression is designated by the pair or operators "?" and ":" when used in the following context:

```
e1 ? e2 : e3
```

which may be interpreted as follows:

```
if "e1" is true
    the value of the expression is "e2"
else
    the value of the expression is "e3"
```

The value of a "true" logical expression is usually non-zero, and the value of a "false" logical expression is usually zero. Logical expressions may be used in arithmetic expressions, with care.

Parentheses may be used, as in BASIC, to force grouping. Brackets are used to indicate array subscripts, rather than the double use of parentheses in BASIC and FORTRAN.

The decrement and increment operators of C look strange initially to BASIC programmers. The increment operator "++" adds one to a variable and the decrement operator "--" subtracts one from a variable. If the operator appears before the variable, the variable is incremented before using its value, and if the operator appears after the variable, the variable is incremented after using its value. For example, the following statement:

```
if (n < 10) ++n;
```

would cause "n" to be incremented by one if its value ware less
than 10, as would the following statement:

```
if (n < 10) n++;
```

since the effect on "n" is the same in both cases. But, in the
following statement:

```
if (n < 10) a[n++] = n;
```

the value of "n" is incremented between uses, storing the value of "n" in a different element of "a[]".

There is an option of the assignment operator "=" analogous to the increment and decrement operators. If the "=" is preceded by one of the following binary operators (call it "b"):

```
+, -, *, /, %, <<, >>, &, ^, |
```

the compound assignment resulting from "b=" in the following expression:

```
u b= expr
```

is equivalent to the following expression:

```
u = (u) b (expr)
```

so that the following (commonly used) expression:

```
w += 2
```

is equivalent to the following expression:

```
w = w + 2
```

in all contexts.

STATEMENTS

The C language has only a limited number of statements, far fewer than BASIC or FORTRAN, since C relies on function libraries to implement all I/O and other system interfaces. Several functions have already been used without explanation to in this tutorial. C uses semicolon to separate statements, whether on the same or on different lines.

The simplest (and sometimes most treacherous) C statement
is the comment statement. It is introduced with "/*" and terminated with "*/". Comments may officially not be nested (according to K and R), although some versions of C allow nested comments. Since C is normally insensitive to end-of-line in many cases, a missing or miscoded "*/" can cause large chunks of a C program to be ignored incorrectly. The best advice on comments is properly to include the termination on the same line as the introduction.

In several cases, expressions may appear as statements. Several instances of this have already been shown. The primary situations in which expressions are used as statements include the following:

- assignment
- function calls
- increment
- decrement
- compound assignment

The "if" statement is somewhat similar to the BASIC "if" statement. Its syntax is as follows:

```
if (expression) statement
if (expression) statement else statement
```

Note that, unlike BASIC, parentheses are required surrounding the control expression. The interpretation of the "if" statement is identical in C and in BASIC. However, the use of Boolean expressions in BASIC may not be compatible with the definitions of TRUE and FALSE in a specific C implementation.

The "while" statement establishes a structured loop in which the control condition is tested before the first (and any subsequent) executions of the body of the loop. Its syntax is as follows:

```
while (expression) statement
```

and it may be represented symbolically by a BASIC sequence similar to the following:

```
100 if (expression) goto 200
    goto 300
200 statement
    goto 100
300 ...
```

The "do" statement establishes a structured loop in which the control condition is tested after the first (and any subsequent) executions of the body of the loop. Its syntax is as follows:

```
do statement while (expression)
```

and it may be represented symbolically by a

BASIC sequence similar to the following:

```
100  statement
     if (expression) goto 100
```

The "for" statement establishes a loop based upon three control expressions. Its syntax is as follows:

```
for (expression1; expression2; expression3)
    statement
```

and it may be represented symbolically by the BASIC sequence similar to the following:

```
     expression1
100  if (expression2) goto 200
     goto 300
200  statement1
     expression3
     goto 100
300  ...
```

Note that the C version of the "for" statement is far more sophisticated than is the BASIC version. Its interpretation is also different, in that it checks the condition the first time, as opposed to BASIC, which does not check the condition the first time.

The "switch" statement provides a multi-way decision scheme similar to an iterated if (...) ... else ...

statement. Its syntax is as follows:

```
switch (expression)
{
    case C1: statement; ....
    case C2: statement; ....
    :
    :
    default: statement; ....
}
```

where the "C1", "C2", etc. represent constants with the same type as the expression and the "default" clause is optional. The expression is evaluated and matched against "C1", "C2", etc. If a match is found, that sequence of statements is executed. If not, the "default" sequence is executed. If a match is found, a "break", "continue", or "goto" is normally required to prevent falling thru to the next case.

The "break" statement causes an immediate exit from the innermost "do", "for", "switch", or "while" statement in which it appears. It functions as if it were a "goto" statement referencing an imaginary label just beyond the end of the statement. Its syntax is as follows:

```
break
```

The "continue" statement causes the next iteration of the innermost "do", "for", or "while" statement to be performed or the loop to be terminated, as the control expression dictates. Its syntax is as follows:

```
continue
```

The "return" statement has already been discussed. It is used to return control to the caller of a function. Its syntax is as follows:

```
return
  or
return (expression)
```

where "expression" provides the value returned by the function, if any. If a value is expected, but none is provided the resulting value is unpredictable. If no value is expected but one is provided, it is discarded.

The "goto" statement is used to explicitly transfer program control to a label. The syntax of a "goto" is as follows:

```
goto label
```

and the syntax of a label is as follows:

```
label: statement (optional)
```

where "label" must follow the rules for C variables and must be defined within the same subroutine or block in which it is used.

PREPROCESSOR

Most C compilers provide a preprocessing capability which extends the compiler's usefulness in a manner similar to that provided by macro assemblers. Lines beginning with "#" are preprocessor definition lines. Although there are many preprocessor definition commands, only the two most common ones are described below.

The most commonly-used preprocessor command is "#define". Its syntax is as follows:

```
#define identifier string
#define identifier (id1,...,idn) string
```

In the first case, subsequent occurrences of "identifier", not in comments or string constants, are replaced by "string". In the second case subsequent occurrences of "identifier" followed by the specified parameter list are replaced by "string", modified by parametric replacement of the "id1" through "idn" in "string". This capability is used primarily for cases such as the following:

```
#define tabsize 2000
:
;
int tab[tabsize];
```

since the size of an array is required to be a constant. Another commonly-used preprocessor command is "include". Its syntax is as follows:

```
#include "filename"
```

This causes the preprocessor to search for "filename" in an operating-system dependent manner, and replace the "include" command with the entire contents of the file, which is, of course, assumed to contain C functions, declarations, statements, etc. Some implementations allow or require alternate syntax of the "#include" statement, primarily concerning the inclusion surrounding the filename.

SUMMARY

This tutorial began on a subset of the C programming language. With this in mind, you should understand enough to start making small programs with help of the C manual and one or two of the many books, written on the subject.

oooooooooo0000000000oooooooooo

File De-fragmentator

by Bob van der Poel
comments by Bob Devries

Here's a nifty little programme which has been supplied to us by Bob van der Poel. The programme is used to unfragment a disk, so that files become more contiguous, that is, they are placed in adjacent sectors on the disk. It is of course most useful on a hard disk, but can be used on floppies as well. This is especially true for those who use the larger format floppies like 80 track ones.

Bob does specify that you must have the 'Kreider C Library' to be able to compile this programme. As usual, this is available from our PD library. This programme will also go into the library for those of you without the necessary C compiler.

Our thanks go to Bob for making this programme freely available. Yet another example of the friendship of OS9 users around the world, even as far as freezing British Columbia!!

Because of the length of this programme, I have had to split it over two issues. So stay tuned for part two!

Regards,
Bob Devries

```
/* File de-fragmentator. This program scans the current or specified
   directory(s) for fragmented files. If found the file is copied using
   pre-extension. Since OS9 attempts to keep all files contigous this
   scheme will work if there is a large enough block on the disk for
   the file in un-fragmented form.

   No attempt is made in this version to de-fragment directory files.

   This program will need the Krieder C library to compile. When compiling
   you will need to give extra memory to the program so that the recursive
   calls don't run out of RAM. "cc unfrag.c -m3k" seems to work alright.

   usage: unfrag [-dr] [directory] [..]
            d - process subdirectories
            r - report only, do not copy files
```

```
            if no directory specified the current directoy will be processed

   Version 1.0, July 1990.
   (c) Bob van der Poel Software
   P.O. Box 57        P.O. Box 355
   Wynndel, BC        Porthill, ID
   Canada V0B 2N0     USA 83853

   This program may be distributed freely for non-commercial use. All
   rights are reserved by the author.
*/

#include <dir.h>
#include <stdio.h>
#include <ctype.h>
#include <direct.h>
#include <errno.h>

#define FALSE 0
#define TRUE  1

#define BUFSIZE 8192    /* buffer size is 32 sectors */

/* Global variables */

direct int dodirs=FALSE;   /* do sub-directory flag */
direct int report=FALSE;   /* report only flag      */
direct char tempfile[20];

char copybuf[BUFSIZE];

direct int totalfls=0;     /* for report at end */
direct int totalfg=0;
direct int totalfix=0;

/* ======================================
   Main...parse command line and set flags,
         call recursive processer.
*/

main(argc,argv)
int argc;
char *argv[];
{

   int doneone=FALSE;
   register int t;
   char *param;
   char tempstr[50];

   setbuf(stdout,NULL); /* unbuffered terminal i/o */
   setbuf(stdin,NULL);

   strcpy(tempfile,"unfrag.XXXXX");  /* create unique filename for copies */
   mktemp(tempfile);
```

```
   while(--argc>0){
      if((*++argv)[0]=='-'){
         for(param=argv[0]+1;*param;param++){
            switch(tolower(*param)){

            case 'd':
               dodirs=TRUE;
               break;

            case 'r':
               report=TRUE;
               break;

            default:
               fputs("Unfrag version 1.0\n",stderr);
               fputs("By Bob van der Poel\n",stderr);
               fputs("July, 1990\n\n",stderr);
               fputs("Usage: Unfrag [-d] [directory] [..]\n",stderr);
               fputs("        d - process sub-directories\n",stderr);
               fputs("        r - report only, do not unfrag\n",stderr);
               quit("");
            }
         }
      }
      else{
        process(*argv);
        doneone=TRUE;
      }
   }

   if(!doneone)  process(".");  /* no directory done, do current */

   /* all done, do report */

   strcpy(tempstr,"\nTotal files checked: ");
   itoa(totalfls,strend(tempstr));
   puts(tempstr);

   strcpy(tempstr,"\nTotal files fragmented: ");
   itoa(totalfg,strend(tempstr));
   puts(tempstr);

   strcpy(tempstr,"Total fixed: ");
   itoa(totalfix,strend(tempstr));
   puts(tempstr);

   exit(0);

}
```

oooooooooo0000000000oooooooooo